Escuela
Politécnica
Superior

Adaptation of neural machine translation systems to domain-specific vocabulary

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Martín García-Ripoll Muñoz

Tutor/es:
Juan Antonio Pérez Ortiz
Felipe Sánchez Martínez

Universitat d'Alacant
Universidad de Alicante

Septiembre 2018

Universidad de Alicante
Escuela Politécnica Superior
Departamento de Lenguajes y Sistemas Informáticos

Adaptation of neural machine translation systems to domain-specific vocabulary

Martín García Ripoll Muñoz

Submitted in part fulfillment of the requirements for the degree of
Graduate in Computer Science of the Universidad de Alicante, September 2018

Abstract

State-of-the-art neural machine translation (NMT) systems are sequence-to-sequence neural networks which take the input sentence in the source language, process it and output a translation in target language. These neural networks can be divided into different components: encoder, decoder and attention mechanism. The goal of this work is to solve the vocabulary size limitation that current NMT systems have and also automate the translation enforcement for some domain-specific words, which means that some words will always need to be translated to other specific words as synonyms do not have the same exact meaning.

The first problem was addressed by adding an extra layer to the neural network, which extracts the embeddings (numeric representations of each word that include information such as meaning, context, etc.) generated by the decoder and the attention matrix for every sentence; this way it was possible to associate each produced unknown word to its embedding and to its source word. The actual translation for that source word is provided using an external method and afterwards it is stored along the embedding in an external dictionary. This way, if anytime later this word appears again, the system would be able to find its embedding close enough to one in the dictionary and will automatically replace the unknown token in the target sentence by the corresponding word in the dictionary.

On the other hand, the second problem also uses a dictionary. However, as it is intended to directly replace a given source word by a specific target word, this dictionary only contains those two words. A modification to the neural network is applied in order to be able to retrieve the attention matrix to align the source and target words so that the system knows which word the NMT system produced for each source word and can therefore easily replace the words contained in the dictionary.

For the experiments, the starting point was a NMT system trained to translate from Catalan into Spanish. For training this neural system, a mix of the Diari Oficial de la Generalitat de Catalunya (DOGC) [1] and El Periodico [2] corpora was used, and for testing the corpus from Consumer Eroski [Alcázar, 2005] was used.

The results obtained during the evaluation of the first problem (the unknown words problem)

[1] http://dogc.gencat.cat/
[2] https://www.elperiodico.com/

show that after translating 10,038 test sentences (with a total of 243,221 words), a total amount of 10,219 unknown words were generated, of which the system proposed was able to replace 58.68%. Moreover, 68% of those replaced words (39.94% of the total unknown words generated) were corrected by the same word that the reference translation uses. As for the BLEU score, the translation provided by the original NMT system scores 61.0 BLEU points while the proposed system gets a score of 64.6. Therefore, from the results of the experiments we can conclude that the addition of this system increased 3.6 BLEU points and helped correcting at least a 39.94% of the unknown words.

The results of the second problem show that out of 255 words, 72% were correctly replaced and 28% were not replaced as expected. This means that using this system around 72% of the domain-specific words are correctly translated, which is a great improvement.

Acknowledgements

I would like to express my sincere gratitude to Juan Antonio Pérez Ortiz, Felipe Sánchez Martínez, the whole Transducens Research Group, my family and friends, specially Lucas Gil Melby.

All of them have helped me during the research process in one way or another. I would like to particularly thank my two TFG supervisors **Juan Antonio** and **Felipe** as they introduced me to the world of machine learning and neural machine translation systems, helped me understand how these architectures work and made me feel as part of the research group.

'Information is the continuity of intrinsic acceptance'

AI Deepak Chopra

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Machine Translation

Translation has been required throughout history since the appearance of the first languages and it has been a key factor in civilizations and globalization. Due to the recent high demand for translation and the high cost of manual translation, some machine translation tools have been developed to help translators in their job. There have been different approaches since the first automatic translation system appeared [Reynolds, 1954], such as rule-based machine translation [Hutchins and Somers, 1992], statistical machine translation (SMT) [Koehn, 2009] or neural machine translation (NMT) [Chalmers, 1990, Chrisman, 1991, Ñeco and Forcada, 1997, Kalchbrenner and Blunsom, 2013, Devlin et al., 2014, Sutskever et al., 2014]. This work intends to try different approaches to solve two problems that the state-of-the-art NMT systems have.

1.2 Neural Machine Translation

Neural machine translation is the newest approach to machine translation which is intended to overtake statistical machine translation systems. Both of these approaches are corpus-based, while rule-based machine translation systems are knowledge-based. The first scientific

papers about NMT were published in the 90s, although there were already some neural systems capable of performing tasks such as speech recognition [Rabiner and Juang, 1993] also used for translation. NMT systems are deep neural networks which receive an input sequence of words in the source language, i.e. a sentence, and output a sequence of words in the target language that has the maximum probability of being the correct translation of the input.

$$p(y_1^I \mid x_1^J) = \prod_{i=1}^{I} p(y_i \mid y_1^{i-1}, x_1^J)$$

where y is the target sentence (the translation) and x is the source sentence (the input), both of them are divided into words where the subindex i is the position of each word in the sentence. I is the length of the target sentence and J is the length of the source sentence. Therefore, y_1^I and x_1^J represent the set of words from the word number 1 to the last word of the target and source sentences respectively; and y_1^{i-1} represents the set of target words from the first one to the last translated word (the previous generated word), i.e. the words that have already been translated at a given time.

As any other neural network, NMT has a first stage of *training*. During this stage, the neural network adjusts its parameters (weights) in order to minimize the error of the results produced. For this task, the neural network is usually provided with some input of which the correct output is known, this way the error can be computed automatically. After various adjustments, the performance of the neural network is tested in a phase called *development* which also utilizes inputs with known outputs but these inputs are different from the training inputs; this way it is possible to detect if the neural network is actually 'memorizing' the example training inputs or if it is really "learning" (a.k.a. overfitting). Depending on the result obtained during this phase, the neural network will keep training or will stop. Whenever the training stops, a final phase called *testing* takes place in order to perform a final evaluation. During this final stage, the neural network utilizes another subset of the corpus with sentences not included neither in the training corpus nor in the development corpus.

The training task for NMT tries to maximize the likelihood of obtaining the best translation

of a given sentence over the rest of possible translations (good and bad ones):

$$\hat{\theta} = \arg\max_{\theta} \sum_{n=1}^{N} \sum_{i=1}^{I_n} \log\left(p(y_i|y_1^{i-1}, x_1^{J_n}; \theta)\right)$$

where θ represents the model parameters, N the number of sentences of the training parallel corpora, y is the target sentence (the translation) and x is the source sentence (the input), both of them are divided into words where the subindex i is the position of each word in the sentence. I_n is the length of the n target sentence and J_n is the length of the n source sentence.

The loss function used to determine how well the neural network is performing is the cross-entropy loss:

$$H(y, \hat{y}) = -\sum_{i=1}^{I} y_i \log(\hat{y}_i)$$

where $\hat{y}$ is the sentence obtained and y is the expected sentence, which are divided into words that are identified by the subindex depending on their position in the sentence.

The neural network parameters are afterwards optimized using stochastic gradient descent, which tries to minimize the loss function in an iterative way [Yin and Kushner, 1997].

During training, a parallel corpus with sentences in source language and their translations in target language is used as the neural network needs a reference for each sentence it tries to translate so it can check its performance. In order to be able to measure the quality of a sentence translation some metrics shall be used. In this work BLEU [Papineni et al., 2002] has been used (it yields a percentage score of success).

Finally, in order to determine when to stop training, a stopping criterium has to be determined. A common approach is to stop when the BLEU score obtained when translating against a small development corpus does not improve more than a given amount of points after a given number of training steps. However, some implementations, like TensorFlow's NMT package, only allow to set as stop criteria a fixed number of training steps.

One of the best architectures proposed for these systems was the encoder-decoder [Cho et al., 2014], as shown in Figure 1.1. It takes the input sentence as a matrix of one-hot encoded vectors, which

means that each word is inputted to the recurrent neural network (RNN) [Jain and Medsker, 1999] as a vector of size $|V|+1$ where V is the vocabulary size[1] and only one of its components is set to 1 (the rest are set to 0) meaning that the inputted word is associated with the position set to 1. Afterwards, the system projects it into a continuous vector of embeddings (1), encodes it generating a compact representation (c) of the source sentence (2), decodes it obtaining a continuous representation of the target sentence (3) and finally converts this representation into the target words (4) [Peris and Casacuberta, 2015]. These compact representations are called states, a new state is created after each input word so the "meaning" of the new word is added to the previous state (each vertical box including c are states). As the last encoder state (c) includes all the previous states, it is considered to be the "sentence state".

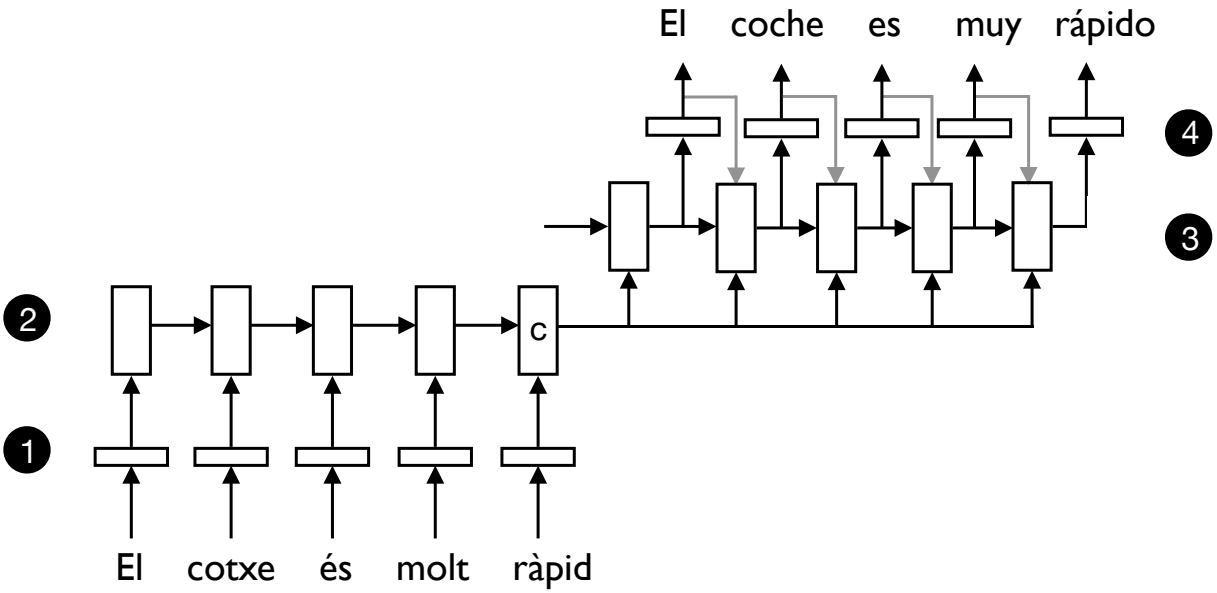


Figure 1.1: encoder-decoder diagram

The embeddings [Hill et al., 2014] are numeric representations of words which are learned during training as any other parameter and interpreted by the neural network for the translation task. Each word in the vocabulary has its own embedding, for example:

$$coche \longleftrightarrow [1.2434, 32.645, 0.2346, 85.129, 36.4, 2.074, ...]$$

[1]One extra position is reserved for the UNK token meaning that the input word is not in the vocabulary and therefore it is unknown.

The issue with this approach is that it does not perform well for long sentences, as the representation of the source sentence (c) is not able to "store" the whole sentence meaning, and therefore the translation will not meet the expected quality. This problem was solved with the addition of an attention mechanism (Figure 1.2) [Bahdanau et al., 2014], which is an extra layer (feed-forward neural network [Svozil et al., 1997]) that decides which source states the decoder has to pay more attention to in order to translate a given word. The first difference with the simple encoder-decoder architecture is that the enconder of this new architecture is (usually) a bidirectional recurrent neural network (Bidirectional RNN), which allows the encoder to generate a compact representation of each source word (1) taking into account the whole sentence context, and not only given the previous words as before. From these states, the attention mechanism determines how important each source word is at a given time step (2), in other words, it determines the importance of each source word associated to each target word. Afterwards, a softmax function (φ) is applied to the weights generated by the attention mechanism. This function normalizes those weights so that their sum is 1.

For example, in the sentence *"El senyal és molt ràpid"*, in order to generate the target word *rápida* the importance of each source word could be 12% for *El*, 8.4% for *senyal*, 3% for *és*, 3% for *molt* and 73.6% for *ràpid*. The vector c_i contains the addition of the product between these values and each state. Note that although *El* and *senyal* might not seem truly related to *rápida*, they help to determine its gender and number. Finally, given the attention vector c_i, the previous decoder state s_{i-1} and the last generated target word, the new decoder state s_i is created (3) and decoded into an embedding which is later turned into a target word. The same way the input words were inputted as one-hot vectors, the output word is determined after applying a Softmax to the score assigned to each word of the target language vocabulary, picking up the one with highest probability-score.

To summarize, the attention mechanism provides a dynamic representation of the relevant part of the source sentence at target step i.

Each attention vector c_i can be considered as a row of the attention matrix. This matrix has been used in order to easily check the alignment of the source and target words (if the first source word has been translated to the second target word) or in other words, to check which

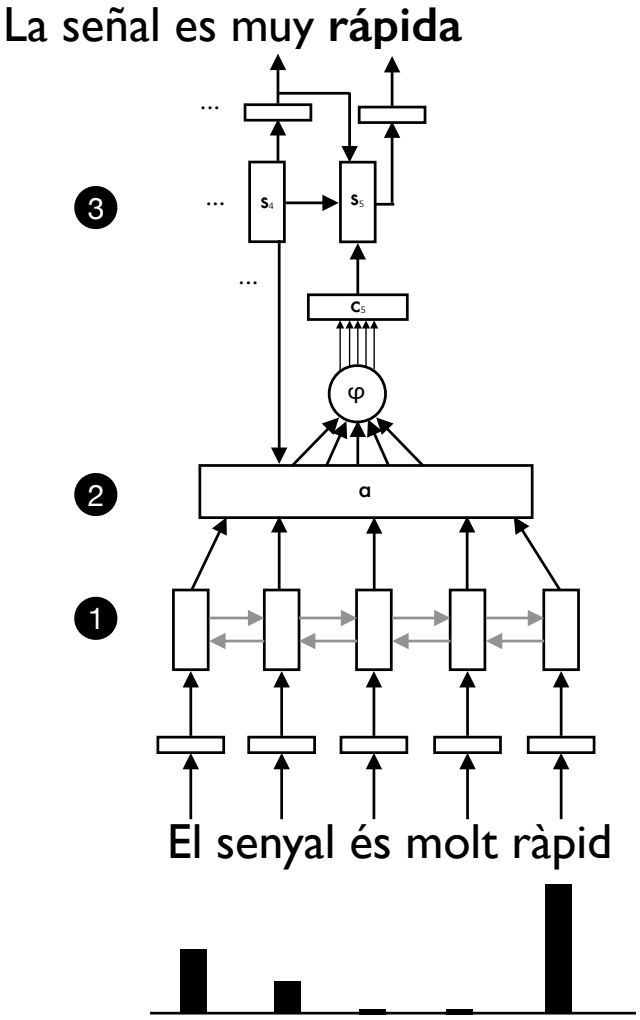


Figure 1.2: Attention Mechanism diagram

source words have influenced the most a given target word. An example of an attention matrix is shown in Figure 1.3.

1.3 Problems addressed and state-of-the-art

The encoder-decoder architecture along with attention mechanism performs well in the translation task, in some cases much better than Statistical Machine Translation [Koehn, 2009, Kinoshita et. al, 2017]. However, it has some limitations that have not been overcome yet.

For the training task, it is required to provide the network with a fixed vocabulary for both the source and target languages. Therefore, if at testing time the source sentence, the target

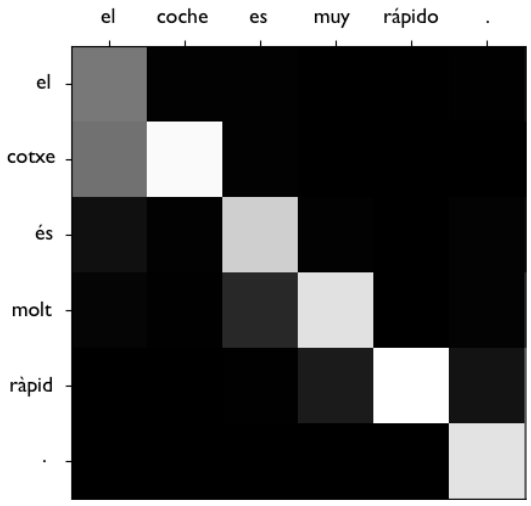


Figure 1.3: Attention Matrix.
The lighter the cell's color, the more attention the system pays to the left word in order to output the top word.

sentence or both of them contain a word not included in that vocabulary, the NMT system will not know which word to output and will produce an `UNK` token, meaning the word to be translated was unknown. The vocabulary needs to be fixed and limited because today's NMT systems cannot handle large vocabularies as it implies storing huge vectors in memory for the input one-hot-vectors and output softmax-vectors, which would turn into a bottle neck (dealing with lots of words is a resource-intensive task and it would drastically slow down the neural network). Moreover, the more parameters the neural network has, the longer the training task will take. A common approach for this problem is byte pair encoding (BPE) [Sennrich et al., 2015], which learns the best way of partitioning the words and processes the corpus applying those partitioning rules. This way, the vocabulary size is greatly reduced as it now contains pieces of words that can be repeated (it is similar to a compression system). BPE also minimizes the chances of obtaining unknown words as the NMT system learns to translate pieces of words into other pieces of words, providing the neural network the "ability" of creating new words by translating a coherent set of pieces of words into other pieces of words that actually do not mean anything when they are put together.

The original NMT system is usually trained with a corpus limited to a specific domain, causing the neural network to perform well in that domain but obtaining poor results when tested

against other domains. A good example of this can be seen when after training with a corpus generated from a newspaper, the NMT system translates a standard sentence to journalistic style sentences. Furthermore, the neural network may encounter some difficulties when translating domain specific words. For example, in a juridic domain, words like *theft*, *robbery* or *burglary* always need to be translated to a specific word as other "synonyms" do not have the exact same meaning. This problem is magnified when these domain-specific terms do not belong to the same domain the neural network was trained for. Nowadays, the only way to solve this problem is editing the translation and manually correcting the domain-specific words.

This work intends to introduce some techniques to overcome or minimize the problem when unknown words appear and cannot be generated and the problem of forcing the system to translate a given word by another concrete word.

Chapter 2

Limited vocabulary and unknown words

As previously stated, the source sentence is split into words, each word is reflected in a one-hot encoded vector and later converted into its associated embedding, which serves as input to the encoder. Later, the decoder outputs the embeddings of the target words that will form the target sentence. Although it might seem that the source and target embeddings are somehow related and that there could exist a correlation between them, there is not. It is not possible to translate a sentence by means of its embeddings, an embedding is a mere representation of the meaning and context in which a given word uses to appear. Every time the decoder outputs an embedding, the NMT system by default checks if there is a close match between that embedding and the embedding of a target word in the target vocabulary.

In order to overcome the limited vocabulary and unknown words problem, the original NMT system needs to be modified. The approach followed in this work consists of adding an external dictionary with the vocabulary that the original system does not have. This dictionary stores the new target words along with their embeddings. Every time a translation is performed, the proposed system will look for any `UNK` tokens generated. Once it has identified all the unknown words and recovered their embeddings, the proposed system will check if the dictionary already contains a close embedding for any of the unknown words. The method used to compute the

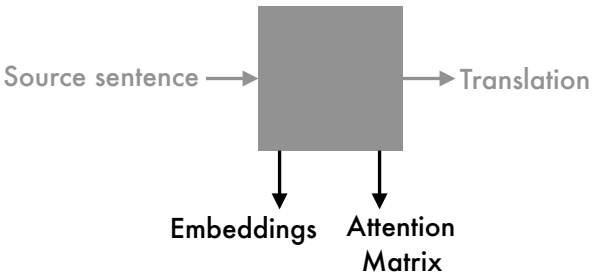


Figure 2.1: NMT system modified as a black-box

closeness of two or more embeddings, is the Euclidean distance. In order to determine whether two embeddings are close enough to replace the unknown token by the dictionary word, a threshold was established after approximately 1,000 tests, which tested the Euclidean distance between same words with same meaning but in different sentences, same words with different meanings (polysemous words), same sentences but using synonyms, etcetera with many different words. Some examples of these tests are shown in Table 2.1.

	agrada	**molesta**	**cotxe**
Same word same meaning	5.89	5.05	5.85
Same word different meaning	7.17	7.66	8.22
Synonyms	6.03	5.95	6.10
Similar words	11.11	9.59	12.65

Table 2.1: Euclidean distance between words. Each number is the average Euclidean distance of at least 5 different sentences.

The experiments resulted in a threshold of 6.0 points of the Euclidean distance, meaning that only embeddings with an Euclidean distance $\leq$6.0 will be considered close enough to make the replacement. If there is only one close match, the associated word in the dictionary will replace the unknown token. In case there is more than one, the closest match will be used. However, if there are no close embeddings in the dictionary, the proposed system will look in the attention matrix for the original word that has been translated into the unknown (see Figure 1.3), it will get a translation for that word in that sentence, either from manual user input or automatically using an alignment model or another automatic translation system, and add the translation and its embedding to the dictionary. If the threshold was higher, some words would be incorrectly

replaced, while if it was lower, there might be some words that would not be replaced and would be left as `UNK` tokens.

The original neural network is modified so that the target embeddings and the attention matrix can be read after the translation. If we looked at the modified NMT system as a black-box, the resulting system would look as shown in Figure 2.1.

In case the unknown word is not in the dictionary yet, the proposed system is designed to ask the user for a translation given the source word and the source sentence.

Some limitations of this approach are not being able to replace sequences of words in the target sentence (so if one word in source language is translated into a sequence of words in target language, the proposed system will not be able to produce that sequence of words, and vice-versa). Another limitation is the lack of automation when providing a translation for an unknown word and the slowdown at translation time if the system is asked for the translation of too many sentences at once, as the current implementation needs to translate the sentences one by one in order to keep track of their attention matrix and their word's embeddings (i.e. the current implementation does not support parallelism).

In order to avoid having to provide manual translation for each unknown word, an external translation system or an external source of bilingual information could be used. Also, the translation speed can be increased by adding some processing to the attention matrix and embedding outputs, which would allow the current implementation to parallelize the translation task as, when parallelizing with TensorFlow, the attention matrices of all sentences are required to have the size of the longest sentence in order to be able to perform mathematical operations efficiently. The processing would mainly consist on cleaning up the matrices that are bigger because of that.

Chapter 3

Domain-specific vocabulary

There are some situations where a given word needs to be always translated into another specific word. This usually happens when translating domain specific texts, especially if they belong to juridical, medical or technical domains.

For this purpose, another modification on top of the original attention-based NMT system is proposed in this work. This modification uses another dictionary, but this time it only stores the source word and its translation.

The only requirement is that the NMT system has to output the attention matrix and the target sentence because the proposed system needs to check the alignments for each target word.

In the first place, this approach uses the neural network to translate the sentence, afterwards it looks for any source word that is included in the domain dictionary and checks in the attention matrix what target words they have been translated into in order to replace them by the specific translation stored in the dictionary.

This proposal has different advantages. On the one hand, not having to force the neural network to learn that a given word should always be translated into another word, which might entail a lower translation quality (e.g. having many repetitions of the same word in the same sentence), and also avoids having to train the whole neural network with a domain-specific corpus every

time you need it to translate texts from a different domain. Furthermore, it does not need any major modifications to the NMT system, so it is easily adaptable to different NMT architectures and implementations.

To summarize, it does not matter the domain the neural network was trained on, the proposed system is an extra component that will automatically edit the translation making sure the domain-specific words are correctly translated.

This proposal presents some limitations such as the incapability of dealing with sequences of words, it only supports one to one translation enforcement (i.e. it can only replace a single word by another word). Also, it presents the same slowdown in translation as explained in Chapter 2. Another additional problem is that the decoder is not aware of the changes produced to the target sentence (the target words replacement), which may result in agreement or concord problems among others. If the decoder is modified so it knows about the changes, it would probably improve the translation quality as the NMT system would translate the following words after the replaced word having information like gender and number of that word that otherwise it would not have.

Chapter 4

Experiments

4.1 Environment

The hardware and software used for this project will be described in this section. In order to train the NMT neural network and perform some experiments, an Amazon Web Service[1] EC2 Instance has been used. The instance was configured as a p2.xlarge instance with 100GB of secondary memory for training and 30 to 50GB for the experiments. For training, it needs more memory as the corpora used takes up a lot of space and the network backs up its state frequently. The p2.xlarge instances have an NVIDIA Tesla K80 GPU, 4 vCPUs and 61 GiB of RAM.

The OS running on the instance was Ubuntu Server 16.04 LTS. Anaconda was installed and configured to run Python 3 with TensorFlow[2] nightly build from 01-10-2018 optimized for GPU usage. In order to be able to use the GPU, CUDA 9.0 [3] and cudNN 7.0 [4] [5] were installed and configured. The NMT package of TensorFlow[6] used is the nightly version from the same

[1] https://aws.amazon.com/
[2] https://github.com/tensorflow/tensorflow/
[3] https://developer.nvidia.com/cuda-90-download-archive
[4] https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v7.0.5/prod/9.0_20171129/Ubuntu16_04-x64/libcudnn7_7.0.5.15-1+cuda9.0_amd64
[5] https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v7.0.5/prod/9.0_20171129/Ubuntu16_04-x64/libcudnn7-dev_7.0.5.15-1+cuda9.0_amd64
[6] https://github.com/tensorflow/nmt

date as TensorFlow. Using nightly versions is recommended; however, working with nightly builds can result in errors or bugs introduced by third parties or changes that would require some modifications of the architecture here proposed. To avoid spending time solving these sort of problems, the versions previously mentioned were fixed and were not updated during development.

4.2 Corpora Used

The corpora used in this work include the Spanish-Catalan corpus from the *Diari Oficial de la Generalitat de Catalunya* (DOGC)[7], *El Periodico*[8] and *Consumer Eroski* [Alcázar, 2005]. All corpora were tokenized, filtered so that every sentence was between 4 and 50 words long, processed so that every number was replaced by a *NUM* token and lowercased after training a TrueCase model [Lita et al., 2003]. The scripts provided by Moses [Koehn, 2009] were used for the tokenization, the filtering and the TrueCasing tasks. The information about the resulting corpora can be found in Table 4.1.

Corpus	**# Sentences**	**# Words CA**	**# Words ES**
DOGC	6,448,644	134,750,251	124,824,033
El periodico	668,955	15,188,660	14,005,575
Consumer Eroski	50,191	1,227,894	1,130,880

Table 4.1: Information about the corpora used in this work.

4.2.1 NMT Baseline-System training corpora

The corpus used for training the NMT baseline-system was a mix of the DOGC and *El Periodico* corpora. Each corpus was divided as shown in Table 4.2.

Finally, those subsets were mixed resulting in the corpora as shown in Table 4.3

The vocabulary for both languages included the 30,000 most common words.

[7] http://dogc.gencat.cat/
[8] https://www.elperiodico.com/

Task	Corpus	%	# Sentences
Training	DOGC	84%	5,416,861
Training	El Periodico	84%	561,922
Development	DOGC	8%	515,891
Development	El Periodico	8%	53,516
Test	DOGC	8%	515,891
Test	El Periodico	8%	53,516

Table 4.2: This table shows how the corpora were divided and the tasks in which they were used.

Task	# Sentences	# Words CA	# Words ES
Training	5,978,783	125,510,036	116,145,028
Development	569,407	12,013,567	11,155,345
Test	569,407	12,049,179	11,122,977

Table 4.3: Summary of the corpora used for each task.

4.2.2 Results evaluation corpora

For the evaluation of the results after the experiments, the corpus from *Consumer Eroski* was used. Note that this corpus was not used for training the neural network, it was used to evaluate the system proposed.

This corpus was divided in two, one for development which was used in the "Limited vocabulary and unknown words" problem, and another part for testing that was used in both problems' evaluation. The development part was used to extract some information such as unknown words from the corpus and later the test part was used to check the performance of the proposed system.

Task	%	# Sentences	# Words CA	# Words ES
Development	80%	40,153	982,319	904,707
Test	20%	10,038	245,575	226,173

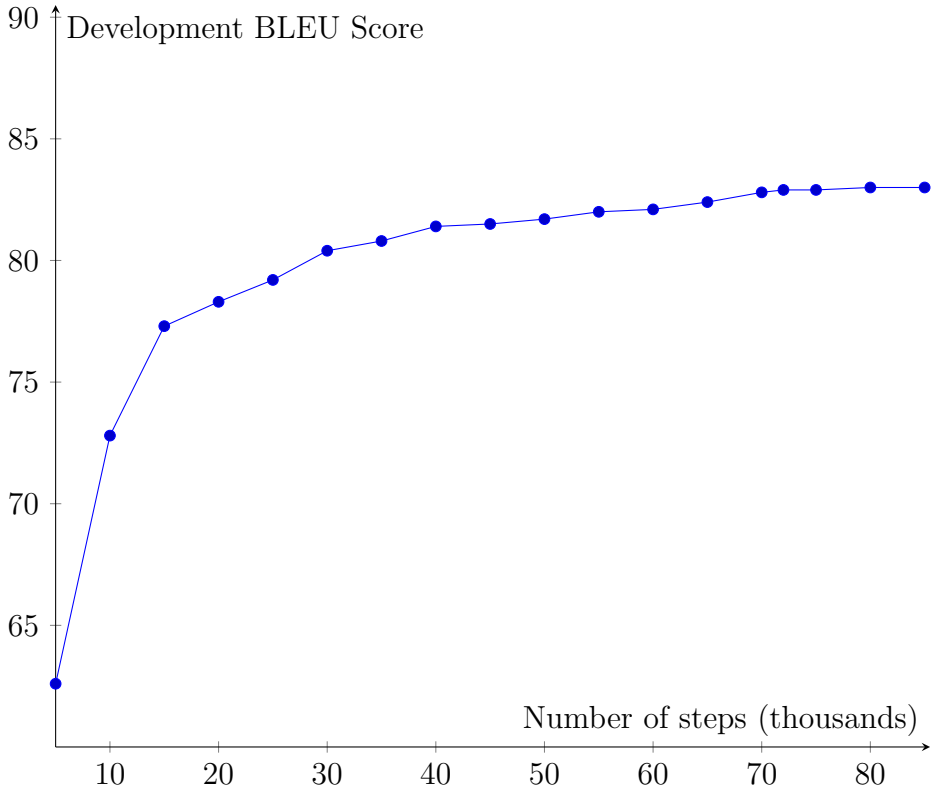
Table 4.4: Consumer Eroski corpus division.

4.3 Neural Network Training

The configuration of TensorFlow was set to use Scaled Luong Attention Mechanism [Luong et al., 2015], the source and target languages were Catalan and Spanish respectively, the number of training

steps was 72,000 and the metric to evaluate the translation performance was BLEU [Papineni et al., 2002]. The neural network was a 2-layer LSTM sequence to sequence model with 128-dim hidden units and embeddings with a dropout of 0.2. Stochastic gradient descent with a learning rate of 1.0 was used.

TensorFlow NMT's implementation only allows to define as stopping criteria a given number of training steps. This number was determined after testing the learning curve of the NMT system for the Catalan-Spanish pair of languages, the results obtained were as follows:



As shown in the graph, the BLEU does not improve much after 65,000 steps and it takes over 1h for each 1,000 steps. This is why the NMT system was trained for 72,000 steps.

The BLEU score and perplexity [Brown et al., 1992] obtained in the last test phase evaluation for this original (non-modified) NMT system were:

- BLEU: 82.9
- Perplexity: 1.56

4.4 Results and Evaluation

4.4.1 Limited vocabulary and Unknown words

Due to the nature of the problem and the system proposed, evaluating the results is not straightforward. The system proposed depends on a dictionary of unknown words and their embeddings. The bigger the dictionary, the better the results. Therefore, running this system with an empty dictionary would behave as if the system was not modified at all, the resulting sentence would be exactly the same as the one generated by the baseline NMT system. On the other hand, if we ran the proposed system with a dictionary containing all the possible words, it would probably outperform the system under normal conditions, which would not be representative either.

In order to obtain representative and reliable results for evaluation, the Consumer Eroski corpus was divided in two parts as explained before, 80% of the corpus was used to generate a dictionary file (development task) using the trained network previously explained. Later, the proposed system translates the remaining 20% of the corpus, this way only the unknown words learned during development will be replaced, leaving the remaining as unknown. The same 20% of the corpus is also translated by the original NMT system, and both translations are compared.

An alignment model [Och et al., 1999] using GIZA++ has been generated using all the corpora here stated in order to obtain the translations of the unknown words and fill the dictionary. Moreover, this alignment model has been used to evaluate the results obtained, checking if the replaced words match the reference words or not. This way it was possible to automatically fill the dictionary and perform an automatic evaluation.

To evaluate the result, the number of unknown words replaced, the number of words correctly replaced, their comparison to the total amount of unknown words and BLEU metrics have been used.

After translating 10,038 sentences (with a total of 243,221 words) of the test evaluation corpus, a total amount of 10,219 unknown words were generated. Once the system proposed was executed

against that translation, it was able to replace 5,997 of the 10,219 unknown words (58.68%), as the remaining words' embeddings had an Euclidean distance higher than 6.0. Moreover, the alignment model shows that 4,082 of those replaced words were replaced by the same word that the reference translation uses. This means that it is safe to affirm that those 4,082 words were correctly replaced but there is not enough information to automatically determine whether the remaining 1,915 replaced words were correct or not, as the system may have used synonyms or other words that would be considered to be valid in that context. These correct words were later classified depending on their Euclidean distance as shown in Table 4.5.

Distance range	**Correct Words**	**Incorrect Words**
[0,1[	2299	1
[1,2[	661	2
[2,3[	372	51
[3,4[	290	357
[4,5[	245	691
[5,6]	215	813

Table 4.5: Relation between Euclidean distance and correctness of the replaced words.

Results in Table 4.5 show that the results get worse as the Euclidean distance increases. In a production environment, it would be possible to set the Euclidean distance threshold to 1, which would greatly decrease the probability of replacing a word incorrectly. However, for this experiment the Euclidean Distance was left as 6.0 because, as explained before, there could be some words classified as *Incorrect words* that are actually correct but do not match the alignment system.

During the manual review of the replacements, it was noticeable that some of the words that were not corrected properly had a "generic embedding" as they are words that do not appear very much in the corpora, and therefore their embeddings have a really broad "meaning". Therefore, the system will replace them even when their embeddings are close not because their meaning in natural language is similar but because the network did not have enough information to give a proper meaning (embedding) to those words and therefore they were assigned a "broad-embedding". For the network, it is probably as if it was replacing the word "something".

As for the BLEU score, the translation provided by the original NMT system on the Consumer Eroski evaluation corpus scores 61.0 BLEU points while the proposed system gets a score of 64.6. Note that the same non-modified system performed 82.9 BLEU points on the DOGC+ElPeriodico corpora and now it performs 60.9 on the experiments corpus. This is due to the different nature and domains of both corpora, and as this network was trained with the first corpora, it performs better when translating in-domain. Moreover, it was also tested the BLEU performance when downgrading the Euclidean distance to 1, resulting in a BLEU score of 64.3.

Therefore, we can conclude that the addition of this system increased 3.6 BLEU points and helped correcting at least a 39.9% of the unknown words properly.

As shown in the examples below, there are a lot of unknown words correctly replaced. However, the last two sentences are clear examples of some of the limitations this system presents: there are some unknown words that cannot be replaced (if they are not in the dictionary yet) and sometimes it can replace a word by an incorrect one. The examples below have been picked from the evaluation corpus.

Original NMT: *para ello se <unk> materiales sólidos.*
Our system: *para ello se empleaban materiales sólidos.*

Original NMT: *en estas bebidas puede haber <unk> de carbono complejos.*
Our system: *en estas bebidas puede haber hidratos de carbono complejos.*

Original NMT: *en otros países europeos, la donación de sangre está <unk> económicamente.*
Our system: *en otros países europeos, la donación de sangre está recompensada económicamente.*

Original NMT: *en la nevera o el <unk>.*
Our system: *en la nevera o el acentuar.*

Original NMT: *entre los más perjudicados destaca el tan <unk> <unk>, que ha visto disminuida la <unk> población en un <num>% en los últimos años.*

Our system: *entre los más perjudicados destaca el tan <unk> avergonzamos, que ha visto disminuida la <unk> población en un <num>% en los últimos años.*

4.4.2 Domain specific vocabulary

In order to test the "Limited vocabulary and Unknown words" proposed system, the Consumer Eroski corpus was used again. This time, a vocabulary containing all the different words was generated and analyzed, selecting the 100 more frequent words of this corpus. Those words were added to the domain dictionary and were to be translated to a '$DOMAIN\_WORD$' token. These 100 words appeared a total of 142,038 times in the corpus, out of the 245,575 words that the corpus has. Note that although not all of those words are domain-specific-words, they were selected in order to have more occurrences and therefore be able to extract representative results and will be referred to as domain-specific-words in this section.

After modifying the original system, trained as explained at the beginning of this chapter, to replace the domain-specific-words included in the domain dictionary, the system was executed and the results were analyzed.

The results show that the words were correctly replaced by the token 105,250 times (74.1%).

Therefore, over 74% of the occurrences of the domain-specific-words contained in the dictionary were correctly replaced, ensuring a specific translation rather than outputting synonyms or other words.

Some of the words included in the domain-specific-vocabulary dictionary are:

cap	anys	ser	pot
molt	aigua	altres	noms
encara	fins	euros	temps

Chapter 5

Concluding Remarks and Future Work

This work is intended to serve as a starting point for future research attempting to solve or minimize the limited vocabulary size problem as well as the domain-specific vocabulary issue. Moreover, it has made me learn many things about machine translation, the state-of-the-art architectures, some of the on-going research works, etc. Furthermore, I learned about TensorFlow, Amazon Web Services and a lot of different tools that I did not know of before. Some of the things I have learned during this work are stated in Appendix A.

The results obtained proof that it is possible to obtain good results following the approach described in this work. The system proposed is able to provide the original neural network with a larger vocabulary even after training, while keeping the coupling with the implementation of the neural network low.

Future work might include the addition of features that allow the system to replace unknown words when there is not a correlation of 1-to-1, as this would probably greatly improve the results, especially for languages that have many compound words, like Germanic languages; also, the evaluation of the domain-specific-vocabulary approach could be performed with only domain-specific words (e.g. using only those words which have a Wikipedia page), which would also require a larger evaluation corpus. Moreover, other methods for determining the threshold for the unknown words problem should be considered. Another possible improvement is the optimization of the translation process while extracting the attention vector and embedding of

each word or the modification of the architecture so that the decoder is aware of the changes produced by the system proposed (when replacing a word), as it was explained in more detail at the end of chapters 2 and 3.

Appendix A

Tasks Performed for this project

Thanks to this work, I have:

- Learned how to use Amazon Web Services (AWS), Amazon's cloud computing platform.
- Created instances of AWS with different configurations.
- Created an image (AMI) of the instance configuration and the environment for this work in AWS.
- Understood the whole architecture of Neural Machine Translation (NMT).
- Understood the current approaches to the problems addressed in this work.
- Learned TensorFlow (Google's open source machine learning framework) which I found pretty difficult due to its complex paradigm.
- Fully understood the implementation of NMT in TensorFlow.
- Learned to process and correctly combine different corpora for training, development and testing.
- Obtained different corpora and processed them.
- Trained NMT systems with different configurations.

- Designed different potential solutions for the problems addressed in this work.
- Implemented them and selected the most promising options.
- Modified the baseline NMT system to extract the attention vectors.
- Modified the baseline NMT system to extract the word embeddings.
- Added extra features that work on top of the neural network.
- Integrated those features with the baseline NMT system.
- Designed an evaluation strategy.
- Evaluated the proposed system.
- among other tasks...

Appendix B

NMT Modifications

This works includes some modifications to the original TensorFlow NMT package.

In order to be able to print the attention matrix and the embeddings, a new file named `tf_print.py` was created. This file includes different functions that print TensorFlow matrices in a clear and legible format which later will be read by the system proposed. These functions need to be called from the neural network at the exact time when the attention matrix is computed and also right before the embeddings are processed. This is why it required making a copy of the original `tensorflow/contrib/seq2seq/python/ops/basic_decoder.py` and `tensorflow/contrib/seq2seq/python/ops/attention_wrapper.py`, to include the print statements without changing the original behavior of the neural network (figures B.1 and B.2).

Finally, the `attention_model.py` and `model.py` files were modified so they call the newly created and modified copies of the previous files (figures B.3 and B.4).

Note that only the modified files are shown in this section, files created for this project can be found in the repository and will not be displayed here.

4 basic_decoder.py

```
28  28    from tensorflow.python.layers import base as layers_base
29  29    from tensorflow.python.ops import rnn_cell_impl
30  30    from tensorflow.python.util import nest
    31  + from . import tf_print
31  32
32  33
33  34    __all__ = [
129 130         `(outputs, next_state, next_inputs, finished)`.
130 131       """
131 132       with ops.name_scope(name, "BasicDecoderStep", (time, inputs, state)):
    133 +       #inputs = Print(inputs, [inputs], message="INPUTS ========> ", summarize=10000000)
132 134         cell_outputs, cell_state = self._cell(inputs, state)
    135 +       # cell_outputs = Print(cell_outputs, [cell_outputs], message="CELL_OUTPUTS ========> ", summarize=10000000)
    136 +       cell_outputs = tf_print.tf_print2file(cell_outputs, [cell_outputs], message="", outputFile="./embeddings_output"
133 137         if self._output_layer is not None:
134 138           cell_outputs = self._output_layer(cell_outputs)
135 139         sample_ids = self._helper.sample(
```

Figure B.1: basic_decoder.py modifications

2 attention_wrapper.py

```
24  24
25  25    import numpy as np
26  26
    27  + from . import tf_print
27  28    from tensorflow.python.framework import dtypes
28  29    from tensorflow.python.framework import ops
29  30    from tensorflow.python.framework import tensor_shape
308 309       g = variable_scope.get_variable(
309 310           "attention_g", dtype=dtype, initializer=1.)
310 311       score = g * score
    312 +   score = tf_print.tf_print2file(score, [score], message="", outputFile="./attention_score.txt")
311 313     return score
312 314
313 315
```

Figure B.2: attention_wrapper.py modifications

10 nmt/attention_model.py

```
21  21
22  22    from . import model
23  23    from . import model_helper
    24  + from . import attention_wrapper
    25  + from . import tf_print
24  26
25  27    __all__ = ["AttentionModel"]
26  28
114 116       # Only generate alignment in greedy INFER mode.
115 117       alignment_history = (self.mode == tf.contrib.learn.ModeKeys.INFER and
116 118                            beam_width == 0)
117     -     cell = tf.contrib.seq2seq.AttentionWrapper(
    119 +     cell = attention_wrapper.AttentionWrapper(
118 120           cell,
119 121           attention_mechanism,
120 122           attention_layer_size=num_units,
148 150
149 151     # Mechanism
150 152     if attention_option == "luong":
151     -     attention_mechanism = tf.contrib.seq2seq.LuongAttention(
    153 +     attention_mechanism = attention_wrapper.LuongAttention(
152 154           num_units, memory, memory_sequence_length=source_sequence_length)
153 155     elif attention_option == "scaled_luong":
154     -     attention_mechanism = tf.contrib.seq2seq.LuongAttention(
    156 +     attention_mechanism = attention_wrapper.LuongAttention(
155 157           num_units,
156 158           memory,
157 159           memory_sequence_length=source_sequence_length,
179 181         tf.transpose(attention_images, [1, 2, 0]), -1)
180 182     # Scale to range [0, 255]
181 183     attention_images *= 255
    184 +   #attention_images = tf.Print(attention_images, [attention_images], message="ATTENTION MATRIX ========> ", summarize=
    185 +   attention_images = tf_print.tf_print2file(attention_images, [attention_images], message="", outputFile="./attention_
182 186     attention_summary = tf.summary.image("attention_images", attention_images)
183 187     return attention_summary
```

Figure B.3: attention_model.py modifications

5 nmt/model.py

```
27  27    from . import model_helper
28  28    from .utils import iterator_utils
29  29    from .utils import misc_utils as utils
    30  + from . import basic_decoder
30  31
31  32    utils.check_tensorflow_version()
32  33
398 399               time_major=self.time_major)
399 400
400 401           # Decoder
401     -         my_decoder = tf.contrib.seq2seq.BasicDecoder(
    402 +         my_decoder = basic_decoder.BasicDecoder(
402 403               cell,
403 404               helper,
404 405               decoder_initial_state,)
450 451                   self.embedding_decoder, start_tokens, end_token)
451 452
452 453             # Decoder
453     -           my_decoder = tf.contrib.seq2seq.BasicDecoder(
    454 +           my_decoder = basic_decoder.BasicDecoder(
454 455               cell,
455 456               helper,
456 457               decoder_initial_state,
```

Figure B.4: model.py modifications

Appendix C

How to run the system

The set up of the system proposed will be explained in this section, but there will not be any explanation about how to train the baseline neural network as it is out of the scope of this document. For more information about how to train a neural network with TensorFlow please refer to the official TensorFlow NMT documentation[1].

The last version of the system used in this work includes:

- NMT package with custom modifications (explained in Appendix B).
- *martingrm* package which includes the main features of this project.

Both packages can be found in GitHub's martingrm/nmt[2] repository.

For the system to properly work, the following requirements must be met:

- At least 10GB of free storage capacity.
- At least 4GB of RAM.
- Python >= 3.6

[1] https://github.com/tensorflow/nmt
[2] https://github.com/martingrm/nmt

- TensorFlow with the optimal configuration for the specific computer. The optimal version for running this system is the nightly version from 01-10-2018, other versions might encounter incompatibilities or unexpected behavior.
- The modified NMT package and the extra packages stated before.

C.1 Instructions

1. Download the code included in the repository and make sure you meet all the requirements to run the system.
2. Create a file with the sentence(s) you want to translate, for better results they should be processed the same way the training corpus was (tokenized, lowercased, etc.).
3. Configure the system depending on which part(s) you want to execute:
   - *Domain-specific vocabulary enforcement* feature: fill the *domain_dictionary.json* file with pairs of words `source:target`.
   - *Unknown words* feature: it is possible to configure the translations input changing the *boolean* variables `manualMode` or `alignMode` to `True` or `False` depending on your interest (if both variables are set to `False` the system will just replace the words it already knows, the ones it already has stored in its dictionary, and will not learn any new unknown words) in the `martingrm/get_unk_emb.py` file.
     - `manualMode` will prompt the user for the translation of any unknown word the system finds while translating.
     - `alignMode` requires an alignment model along with the source and reference sentences so the system can automatically extract the translation of each unknown word.
4. Finally, you only have to run the following command inside the *nmt/* folder:

```
python3 run.py \
--out_dir=/path/to/trained/neuralnetwork    \
```

```
--inference_input_file=/path/to/input/sentences/file.lang    \
--inference_output_file=/path/to/output/sentences/file.lang
```

It is also possible to add the following options in order to tell the system which parts you want to execute:

```
--unknown=True --domain=True
```

I strongly recommend redirecting the output to another file in case you want to review the logs later.

C.2 Example

We will translate the sentence "`ahir em van furtar la cartera a la feina .`". Note that it has been tokenized and lowercased as all the corpora used in this work.

Let's assume the system will translate `furtar` by `robar` but we want it to translate it by `hurtar`. We need to configure the system so it makes sure the word *furtar* is correctly translated. To do so, we need to add the following to the file `domain_dictionary.json`:

```
{
...
"furtar":"hurtar",
...
}
```

Also, we realize that the word `ahir` was not in the neural network vocabulary and we want the system to learn it. For this purpose, we set the option `manualMode` to `True` in the `martingrm/get_unk_emb.py` file (and leaving the option `alignMode False`).

Assuming:

- we stored the sentence in a single file in `/users/ubuntu/input.ca`
- we stored the previously trained neural network in `/users/ubuntu/nn/`
- we want to save the output to `/users/ubuntu/translation.es`

We can run the system from inside the `nmt/` folder as follows:

```
python3 run.py \
--out_dir=/users/ubuntu/nn    \
--inference_input_file=/users/ubuntu/input.ca    \
--inference_output_file=/users/ubuntu/translation.es
```

The system will:

1. Run the neural network to translate the sentence.
2. Replace the word `robar` (that the neural network translated incorrectly) by `hurtar`.
3. Ask the user for the translation of `ahir`.
4. And finally, once the user provides this translation, it will automatically use it for the current sentence, saving the final translation in `/users/ubuntu/translation.es`.

Bibliography

[Alcázar, 2005] Asier Alcázar. "Towards linguistically searchable text". *In Proceedings of BIDE (Bilbao-Deusto) Summer School of Linguistics 2005*, Bilbao. Universidad de Deusto. 2005.

[Bahdanau et al., 2014] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". arXiv preprint arXiv:1409.0473. 2014.

[Brown et al., 1992] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Jennifer C. Lai and Robert L. Mercer. "An Estimate of an Upper Bound for the Entropy of English". *Computational Linguistics, vol. 18, no. 1, pp. 31-40.* 1992.

[Chalmers, 1990] Chalmers, D.J. "Syntactic Transformations on Distributed Representations", Connection Science 2:12, 5362. 1990.

[Cho et al., 2014] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau and Yoshua Bengio. "On the properties of neural machine translation: Encoder-decoder approaches". arXiv preprint arXiv:1409.1259. 2014

[Chrisman, 1991] Chrisman, L. "Learning Recursive Distributed Representations for Holistic Computation", *Connection Science 3:4, 345366.* 1991.

[Devlin et al., 2014] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz and John Makhoul. "Fast and robust neural network joint models for statistical machine translation". *In 52nd Annual Meeting of the Association for Computational Linguistics*, Baltimore, MD, USA. June, 2014.

[Hill et al., 2014] Felix Hill, Kyunghyun Cho, Sebastien Jean, Coline Devin, and Yoshua Bengio. "Embedding word similarity with neural machine translation". arXiv preprint arXiv:1412.6448. 2014.

[Hutchins and Somers, 1992] Williams John Hutchins and Harold L. Somers. "An introduction to machine translation". *London Academic Press*. ISBN 0-12-362830-X. 1992.

[Jain and Medsker, 1999] Lakhmi C. Jain and Larry R. Medsker. *Recurrent Neural Networks: Design and Applications*. ISBN: 0849371813. 1999.

[Kalchbrenner and Blunsom, 2013] Nal Kalchbrenner and Phil Blunsom. "Two recurrent continuous translation models". *In Proceedings of the ACL Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics. 2013.

[Kinoshita et. al, 2017] Satoshi Kinoshita, Tadaaki Oshio and Tomoharu Mitsuhashi. "Comparison of SMT and NMT trained with large Patent Corpora: Japio at WAT2017". *Proceedings of the 4th Workshop on Asian Translation, pages 140145*, Taipei, Taiwan. 2017.

[Koehn et al., 2007] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin and Evan Herbst. "Moses: Open Source Toolkit for Statistical Machine Translation". *In Proceedings of the ACL-2007 Demo and Poster Sessions, pages 177180*, Prague, Czech Republic. Association for Computational Linguistics. 2007.

[Koehn, 2009] Philipp Koehn. "Statistical Machine Translation". *Cambridge University Press. p. 27.* ISBN 0521874157. Retrieved 22 March 2015. Statistical machine translation is related to other data-driven methods in machine translation, such as the earlier work on example-based machine translation. Contrast this to systems that are based on hand-crafted rules. 2009.

[Lita et al., 2003] Lucian Vlad Lita, Abe Ittycheriah, Salim Roukos and Nanda Kambhatla. "tRuEcasIng". *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Sapporo, Japan. pp. 152159. 2003.

[Luong et al., 2015] Minh-Thang Luong, Hieu Pham and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". arXiv preprint arXiv:1508.04025. 2015.

[Ñeco and Forcada, 1997] Ramón P. Ñeco and Mikel L. Forcada. "Asynchronous translations with recurrent neural nets". *In International Conference on Neural Networks, volume 4, pages 2535 2540*. 1997.

[Och et al., 1999] Franz Josef Och, Christoph Tillmann and Hermann Ney. "Improved alignment models for statistical machine translation". *Proc. of the Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora*. 1999.

[Papineni et al., 2002] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. "BLEU: a method for automatic evaluation of machine translation". *ACL-2002: 40th Annual meeting of the Association for Computational Linguistics. pp. 311318*. CiteSeerX 10.1.1.19.9416. 2002.

[Peris and Casacuberta, 2015] Álvaro Peris and Francisco Casacuberta. "A bidirectional recurrent neural language model for machine translation". *Procesamiento del Lenguaje Natural*. 2015.

[Rabiner and Juang, 1993] Lawrence Rabiner and Biing-Hwang Juang. "Fundamentals of speech recognition", Prentice-Hall, Inc., Upper Saddle River, NJ. 1993.

[Reynolds, 1954] A. Craig Reynolds. The conference on mechanical translation. Mechanical Translation. 1954.

[Sennrich et al., 2015] Rico Sennrich, Barry Haddow and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units". arXiv preprint arXiv:1508.07909. 2015.

[Sutskever et al., 2014] Ilya Sutskever, Oriol Vinyals and Quoc V. Le. "Sequence to sequence learning with neural networks". arXiv preprint arXiv:1409.3215. 2014.

[Svozil et al., 1997] Daniel Svozil, Vladimír Kvasnicka and Jiří Pospichal. "Introduction to multi-layer feed-forward neural networks". 1997.

[Yin and Kushner, 1997] George Yin and Harold J. Kushner. “Stochastic Approximation and Recursive Algorithms and Applications”. 1997.